|  |  |
| --- | --- |
|  | **Imaging**  **Software Development Kit** |

|  |  |
| --- | --- |
| Purpose | Presents the U8500 Imaging Software Development Kit.  Details the development environment, Integrating a new camera/sensor |
| Document Status | Draft |
| Date | March 12, 2012 |
| Document  Version | V0.1 |

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Release Date | Writer | Update |
| V0.1 |  | Chetan Nanda | Creation – Initial Draft |

Legal Information

**© Copyright ST-Ericsson, 2009. All Rights Reserved.**

**Disclaimer**

The contents of this document are subject to change without prior notice. ST-Ericsson makes no representation or warranty of any nature whatsoever (neither expressed nor implied) with respect to the matters addressed in this document, including but not limited to warranties of merchantability or fitness for a particular purpose, interpretability or interoperability or, against infringement of third party intellectual property rights, and in no event shall ST-Ericsson be liable to any party for any direct, indirect, incidental and or consequential damages and or loss whatsoever (including but not limited to monetary losses or loss of data), that might arise from the use of this document or the information in it.

ST-Ericsson and the ST-Ericsson logo are trademarks of the ST-Ericsson group of companies or used under a license from STMicroelectronics NV or Telefonaktiebolaget LM Ericsson.

All other names are the property of their respective owners.

**Trademark List**

All trademarks and registered trademarks are the property of their respective owners.

In addition to the generic statement above, please fill in all trademarks and registered trademarks that could be identified.

See examples below.

|  |  |
| --- | --- |
| *Microsoft®* | Microsoft is a registered trademark of Microsoft  Corporation in the United States and/or other  countries. |
| *OBEX™* | OBEX is a trademark of Infrared Data Association. |
| *Android™* | Android is a trademark of Google Inc*.* |
| *Ubuntu®* | Ubuntu is a registered trademark of Canonical. |
| *Sony®* | Sony is a registered trademark of Sony Corp. |

Table of Contents

[1. References 6](#_Toc320882972)

[2. Acronyms and Terms 7](#_Toc320882973)

[2.1 Glossary 7](#_Toc320882974)

[3. Introduction 8](#_Toc320882975)

[3.1 STE Imaging stack overview 8](#_Toc320882976)

[3.2 Scope 8](#_Toc320882977)

[3.3 Audience 8](#_Toc320882978)

[4. Environment Setup 9](#_Toc320882979)

[4.1 SW Environment 9](#_Toc320882980)

[4.1.1 SW Extraction and Build 9](#_Toc320882981)

[4.1.2 Flashing Process 10](#_Toc320882982)

[4.1.3 Building individual modules 10](#_Toc320882983)

[4.1.4 Building ISP FW: 10](#_Toc320882984)

[4.1.5 Imaging shared libraries 12](#_Toc320882985)

[4.2 HW Environment 12](#_Toc320882986)

[4.2.1 HW Requirements 12](#_Toc320882987)

[4.2.2 Setup 13](#_Toc320882988)

[13](#_Toc320882989)

[5. Integrating new Sensor 16](#_Toc320882990)

[5.1 Changes at Android/OMX 16](#_Toc320882991)

[5.2 Changes at LLCD 18](#_Toc320882992)

[5.2.1 LLCD Directory Structure 19](#_Toc320882993)

[5.2.2 Sensor Files 20](#_Toc320882994)

[5.2.3 AF Driver Files 24](#_Toc320882995)

[5.2.4 OTP/NVM Detail 24](#_Toc320882996)

[5.3 MMIO Driver 25](#_Toc320882997)

[5.3.1 XSHUTDOWN Signal. 25](#_Toc320882998)

[5.3.2 ExtClk frequency 26](#_Toc320882999)

[6. Validation 27](#_Toc320883000)

[6.1 ISP FW Validation 27](#_Toc320883001)

[6.2 OSI Components Validation 27](#_Toc320883002)

[7. Open Points 28](#_Toc320883003)

# 1. References

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Category** | **Title** | | **Version** | **Editors** |
| User Guide | /1/ | Getting Started with Android And Linux |  |  |
| SAS | /2/ | U8500 Linux Imaging Sub-System Software Architecture Specification | V1.0 | Preetika BHASIN/  Eric AUGER/  PY Taloud |
| SW | /3/ | U8500 Compiling STxP70 ISP FW Code | V1.0 | Chetan NANDA |
| SW | /4/ | U8500 LLCD Documentation.chm |  |  |
| SW | /5/ | TestCase\_description\_ite\_nmf.doc | V1.2 | FW Team |
| SW | /6/ | U8500 Multimedia Test Environment | V1.0 | Validation Team |
| TEST | /7/ | ‘Camera\_TestPlan\_OSI\_wk12.xlsm |  | Validation Team |

# 2. Acronyms and Terms

## 2.1 Glossary

|  |  |
| --- | --- |
| **Term** | **Definition** |
| 3A | AEC + AWB + AF |
| ExtClk | External Clock |
| FW | Firmware |
| GB | Android’s Gingerbread |
| HAL | Hardware Abstraction Layer |
| HREF | STE Reference Board |
| ICS | Ice Cream Sandwich Android Release |
| ISL | Intelligent Status Line |
| ISP | Image Signal Processor |
| ITE | Integrated Test Environment |
| LLCD | Low Level Camera Driver |
| MMDSP | Multimedia DSP |
| MMIO | Multimedia Input Output |
| MMTE | Multimedia Test Environment |
| NMF | Nomadik Multimedia Framework |
| NVM | Non Volatile Memory |
| OMX | OpenMax |
| OTP | One Time Programmable |
| SDK | Software Development Kit |
| SI | System Integration |
| SMIA | Standard Mobile Imaging Architecture |
|  |  |

# 3. Introduction

Imaging SDK provide information about STE Imaging development environment. It provides details for:

* Setting up SW environment.
* Setting up HW environment.
* Steps for integrating a new sensor (RAW Bayer) in imaging stack.
* Validation/Testing environment (MMTE/ITE-NMF).

This version based on Android’s ICS release and may require update for GB release.

## 3.1 STE Imaging stack overview

ST-Ericsson’s imaging SW stack run on three different processing units.

|  |  |
| --- | --- |
| STxP70 | ISP Firmware + LLCD |
| MMDSP | NMF/ISPCTL Components |
| ARM | OMX Components, Kernel Drivers, Android HAL, and SW Algorithms. |

For more information on SW split refer to /2/.

Integration of new sensor in imaging stack majorly required changes at LLCD, OMX and Android HAL.

In coming sections we will cover those changes in details taking Sony’s IMX072 as a reference sensor.

## 3.2 Scope

This document covers the following:

* Setting up of Android environment using SI release **5.49**.
* Build/Debug individual OMX components.
* Modify/Build ISP Firmware (FW).
* Integrating a new sensor.

## 3.3 Audience

The audience of this document is limited to engineers/developers working on STE U8500 Imaging platform.

# 4. Environment Setup

Environment setup involves, setting up software and hardware environment.

Note: Follows steps are tried in STE infrastructure and may need bit of modification in adapting these to customer’s environment.

## 4.1 SW Environment

Software environment setup includes:

* Extracting code base
* Compiling the complete code base.
* Flashing image onto target (U8500 based SoC)

More information about building and flashing Android image can be found in /1/

### 4.1.1 SW Extraction and Build

STE HREF releases (also called SI releases from now on) are available on STE internal code server ‘Gerrit’

Following commands are used to sync code from Gerrit server:

|  |
| --- |
| #Sync code from Gerrit server  $repo init -u ssh://$USER@gerrit.lud.stericsson.com:29418/manifests/u8500/android -b refs/tags/u8500-android-4.0\_v5.49  $repo sync |

Above two commands will create an Android environment.

Here ‘u8500-android-4.0\_v**5**.**49**’ is a tag for SI5.49 release.

Imaging validation environment MMTE and ITE-NMF are compiled out by default. Following modification has to be done to include them in build:

|  |
| --- |
| In file ‘vendor/st-ericsson/multimedia/linux/build/Dirs.u8500.mk’  Make sure 'BUILD\_VALID’ and ‘BUILD\_ITE\_OMX\_USE\_CASES’ must be set to ‘true’  BUILD\_ITE\_OMX\_USE\_CASES=true  BUILD\_VALID = true |

Following are commands to be used for compiling the whole environment:

|  |
| --- |
| #Compiling  **$ bash**  **$ source build/envsetup.sh**  including vendor/st-ericsson/build/vendorsetup.sh  Verifying platform development build environment  including vendor/st-ericsson/multimedia/linux/build/mmbuild.sh  **$ lunch**  You're building on Linux  Lunch menu... pick a combo:  1. full-eng  2. full\_x86-eng  3. vbox\_x86-eng  4. ste\_u8500-eng  5. ste\_u8500-user  6. ste\_u8500\_hats-eng  Which would you like? [full-eng] **4** <Select ste\_u8500-eng >  **$ make**  …. This will compile the complete code … |

More information about building and flashing Android can be found in /1/

### 4.1.2 Flashing Process

Successful compilation will generate:

Binaries in <path>/out

Flash Image in <path>/images

|  |
| --- |
| # Flash procedure  From <path>/images/ste\_u8500\_android\_mop500\_power\_defconfig, run following command  flasher.bat -t hrefp\_v20\_v10\_db8500b0\_secst -L -e –c  Where ‘hrefp\_v20\_v10\_db8500b0\_secst’ is a target |

More detail about ‘flasher’ can be found at: <path>\images\ste\_u8500\_android\_mop500\_power\_defconfig\readme-flasher

### 4.1.3 Building individual modules

Following procedure to be followed for building an individual component in Android environment

|  |
| --- |
| # Compiling individual component  $ cd <path>/multimedia/linux/build  $ mm DIRS\_TO\_BUILD=imaging/xxx clean all  Where ‘xxx’ is a imaging component, e.g. to build hsmcamera use following command:  $ mm DIRS\_TO\_BUILD=imaging/hsmcamera |

### 4.1.4 Building ISP FW:

ISP FW is running on STxP70 micro-controller and needs STxP70 tool chain for compilation.

ISP FW source code is included in the SI release at:

<path>/vendor/st-ericsson/multimedia/imaging/isp8500\_firmware\_XXX/source/FW\_src

Where XXX – Sensor Name. E.g. isp8500\_firmware\_IMX072 for STE reference sensor.

Currently STxP70 tool chain is not integrated in SI releases. So ISP FW source code is not compliable under Android environment. For ISP FW Build refer /3/.

This section provides how to change ISP FW in Android release.

ISP FW compilation generates

Four binaries under ‘isp\_fw\Binaries’:

* Isp8500\_primary\_fw\_data.bin
* Isp8500\_primary\_fw\_ext.bin
* Isp8500\_primary\_fw\_ext\_ddr.bin
* Isp8500\_primary\_fw\_split.bin

Three headers file under ‘isp\_fw\DeviceParams’:

* baseline.h
* extension.h
* nvm.h

Integrating new ISP FW binaries is two-step process, first ISP FW binaries are integrated to generate a re-locatable library ‘*libst\_isp8500\_firmware\_primary.a*’. This (generated) library is then linked to shared library of ‘imgcommon’ component ‘*libste\_ens\_image\_common.so’*

Following are the steps to integrate ISP FW binaries in Android build:

|  |
| --- |
| #Integrate ISP FW binaries in Android build   1. Copy ISP FW binaries i.e. Isp8500\_primary\_fw\_data.bin, Isp8500\_primary\_fw\_ext.bin, Isp8500\_primary\_fw\_ext\_ddr.bin, Isp8500\_primary\_fw\_split.bin   To ‘<path>/vendor/st-ericsson/multimedia/imaging/isp8500\_firmware\_XXX/isp8500\_binaries’   1. Copy ISP FW headers i.e. baseline.h, extension.h nvm.h   To ‘<path>/vendor/st-ericsson/multimedia/imaging/isp8500\_firmware\_XXX/include’   1. Change to dir ‘<path>/vendor/st-ericsson/multimedia/imaging/isp8500\_firmware\_XXX/isp8500\_binaries’ 2. Run ‘genall.sh’   *./genall.sh*   1. Change to dir ‘<path>/vendor/st-ericsson/multimedia/linux/build’ 2. Build ‘isp8500\_firmware\_XXX’   *mm DIRS\_TO\_BUILD=imaging/isp8500\_firmware\_XXX clean all*  This will generate *‘libst\_isp8500\_firmware\_primary.a’*   1. ISP FW re-locatable library is linked into ‘imgcommon’, so we need to build imgcommon also.   *mm DIRS\_TO\_BUILD=imaging/imgcommon clean all*  This will generate a *‘libste\_ens\_image\_common.so’,* This .so is then pushed to target at  ‘*/system/lib’* |

### 4.1.5 Imaging shared libraries

STE Imaging SW stack consist of Android HAL, OMX Components and ISP FW. For changing any of the components on flashed image, we just need to push (adb command) the shared library of corresponding component.

**Android HAL:**

Generate *libcamera.so*, and can be modified by pushing a new libcamera.so at ‘*/system/lib*’

**OMX Components:**

OMX Components generate following shared libraries:

HSMCamera (*libste\_camera.so*) - target position ‘*/system/lib/ste\_omxcomponents*‘

Re-locatable libraries for ‘sw3a, tuningloader’ etc… are linked into this shared library.

Imgcommon(*libste\_ens\_image\_common.so*) – target position ‘*/system/lib’*

Re-locatable libraries for ‘ISP FW, ifm’ etc… are linked into this shared library.

Ispproc (‘*libste\_ens\_ispproc.so*’) - target position ‘*/system/lib/ste\_omxcomponents*‘

## 4.2 HW Environment

This section provides details regarding required hardware and setting them together.

Steps provided are for STE HREF environment and might need adaptation while working with customer board.

### 4.2.1 HW Requirements

Following HW are needed while working with STE Imaging development:

* Windows machine - Preferably Windows XP and later.
* Linux machine – Preferably Ubuntu 10.04 and later.
* JTAG Debugger (Lauterbach).
* Power cable for Lauterbach (Output – 7.5V, 1.7A).
* Board - U8500.
* Power cables for the board (Output – 4V, 2.5A).
* Serial cable.
* USB cable (Type A to Type B) to connect Lauterbach to Test PC (Windows).
* ARM20 to MIPI converter.

### 4.2.2 Setup

Following describe the steps for setting up the hardware.

|  |
| --- |
|  |

**Complete Setup**

**Step1:**

Lauterbach is used for source level debugging of code running on ARM core. Its also useful while debugging MMDSP core dumps.

Connect u8500 board to Lauterbach using a debug cable and ARM20 to MIPI converter.

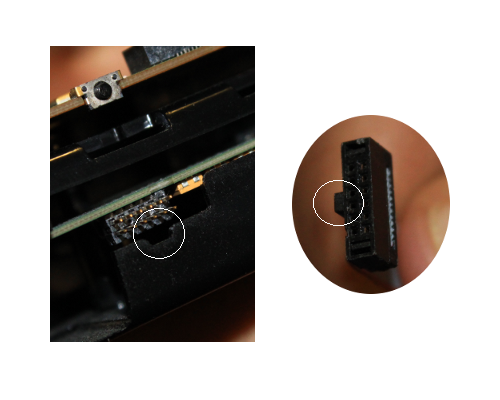
The white dot and the broken pin should be aligned properly as shown in the picture below. The red line on the bus cable should be towards left.

|  |
| --- |
|  |

**Step 2:**

Connect u8500 board using a serial cable to the windows PC.

Check for the circled notches (check the following image) on the board and serial connector. These notches should be aligned properly. The red line on the bus cable should be towards left.



**Step 3:**

Lauterbach should be connected to the windows PC using USB cable (Type A to Type B).

**Step 4:**

Then connect the power cables to Lauterbach and u8500.

**Note:** Once the Lauterbach is powered on, connect the power cable to the u8500 board and power on it. Any change in the sequence may damage the Lauterbach.

**Step 5:**

**Verifying the test setup:**

Powering on the board will display ST Ericsson logo on the LCD screen on the board and two LED will start just below the LCD.



# 5. Integrating new Sensor

In STE Imaging SW stack, sensor specific change/code has been well structured and well abstracted. This results in defined interfaces that abstract sensor specific change from rest of the generic code.

This section provides details about integrating a new sensor (RAW Bayer) into STE Imaging stack. This includes:

* Adding Support for a new sensor in LLCD.
* Adding Support for new sensor in Android Build.

## 5.1 Changes at Android/OMX

In STE Imaging SW stack there are two dedicated components for each supported sensor located at <path>/vendor/st-ericsson/multimedia/imaging/

* isp8500\_firmware\_xxx
* tuning\_xxx

Where ‘xxx’ is sensor name e.g. IMX072 for STE reference sensor.

Isp8500\_firmware\_xxx: Contains LLCD + ISP FW code for a sensor.

Tuning\_xxx: Contains Tuning data for sensor.

Adding a support for a new sensor required changes mainly in LLCD and few changes needed in OMX and HAL code.

This section provides details for adding a new sensor in Android build system using IMX072 as a reference example.

|  |
| --- |
| #New sensor integration   1. Copy sensor specific components at ‘<path>/vendor/st-ericsson/multimedia/imaging/’   e.g.  isp8500\_firmware\_IMX072  tuning\_IMX072   1. Modify platform specific ‘*BoardConfig.mk*’ for selecting a new senor for build.   BoardConfig.mk is present at *‘/vendor/st-ericsson/u8500’*  Under ‘#select camera sensor’, set the primary sensor as given below:  **# Select Camera Sensor**  **CAMERA\_SET\_PRIMARY\_SENSOR ?= IMX072**   1. Modify ‘*Dirs.u8500.mk’* for including sensor specific firmware and tuning data into build system.   Dirs.u8500.mk present at ‘*vendor/st-ericsson/multimedia/linux/build/*’  ifeq ($(BUILD\_IMAGING),true)  REF\_DIRS\_TO\_BUILD += \  …  **+ifeq ($(CAMERA\_SET\_PRIMARY\_SENSOR),IMX072)**  **+REF\_DIRS\_TO\_BUILD += \**  **+imaging/isp8500\_firmware\_IMX072 \**  **+imaging/isp8500\_firmware\_IMX072\_secondary \**  **+imaging/tuning\_IMX072**  **+endif #ifeq ($(CAMERA\_SET\_PRIMARY\_SENSOR),IMX072)**  …  endif   1. Modify ‘*Vars.u8500.mk*’ to define an ‘IMG\_CONFIG’ flag for a new Sensor/Program.   Vars.u8500.mk present at ‘*vendor/st-ericsson/multimedia/linux/build/*’  **ifeq ($(CAMERA\_SET\_PRIMARY\_SENSOR),IMX072)**  **IMG\_CONFIG=201**  **else**  **IMG\_CONFIG=100**  **endif**  IMG\_CONFIG value 201 is for IMX072 sensor. Similarly this list can be extended for any new sensor.  On building the code a file ‘*ImgSelectedConfig.h*’ would be generated at ‘/vendor/st-ericsson/multimedia/imaging/imaging\_platform\_configuration/include’ which specify IMG\_CONFIG value for currently selected sensor.   1. Define Sensor/Program specific configuration in ‘*ImgConfig.h*’   ImgConfig.h is present at ‘/vendor/st-ericsson/multimedia/imaging/imaging\_platform\_configuration/include’  This file contains configurations that are sensor/program specific and must be set properly.  **+#if (IMG\_CONFIG == 201)**  **+ #define IMG\_CONFIG\_PRIMARY\_SENSOR 2**  **+ #define IMG\_CONFIG\_SECOND\_SENSOR 4**  **+ #define IMG\_CONFIG\_UI 1**  **+ #define IMG\_CONFIG\_FLASH 1**  **+ #define IMG\_CONFIG\_CUSTOMER\_FEATURE 2**  **+ #define IMG\_CONFIG\_EXIF\_MAKER\_NOTES IMG\_CONFIG\_FALSE**  **+ #define IMG\_CONFIG\_NVM\_FROM\_FILE IMG\_CONFIG\_FALSE**  **+ #define IMG\_CONFIG\_PREFLASH\_USING\_TORCHMODE IMG\_CONFIG\_FALSE**  **+ #define IMG\_CONFIG\_GSAMPLE\_TUNING\_ALGO IMG\_CONFIG\_FALSE**  **+ #define IMG\_CONFIG\_SW\_PLATFORM 1**  **+ #define IMG\_CONFIG\_SENSORNVM\_READ 2**  **+#endif**  These defines decide e.g. Whether NVM data present in sensor module, Sensor driven flash to be used or not etc …   1. Changes in Android HAL 2. Modify ‘*Android.det*’ present at ‘*vendor/st-ericsson/hardware/libcamera*’   Add entry for IMX072 sensor  **+# check for IMX072 sensor**  **+ifeq (IMX072,$(filter IMX072%,$(CAMERA\_SET\_PRIMARY\_SENSOR)))**  **+MY\_PRIMARY\_SENSOR := IMX072**  **+LOCAL\_CFLAGS += -DSTE\_SENSOR\_IMX072=1**  **+endif #module configuration IMX072 sensor**   1. Modify ‘*Android.opt’* present at ‘*vendor/st-ericsson/hardware/libcamera*’   Add entry for IMX072 sensor, specifying the sensor sizes.  **+ifeq ($(strip $(MY\_PRIMARY\_SENSOR)),IMX072)**  **+LOCAL\_CFLAGS += -DPRIMARY\_RAW\_BAYER=1**  **+LOCAL\_CFLAGS += -DDEFAULT\_PICTURE\_WIDTH=2592**  **+LOCAL\_CFLAGS += -DDEFAULT\_PICTURE\_HEIGHT=1944**  **+endif # MY\_PRIMARY\_SENSOR** |

## 5.2 Changes at LLCD

ISP FW is responsible for controlling/configuring ISP and Sensor.

ISP FW is structured in a way such that changes required to integrate a new sensor are abstracted by set of low level apis (LLA). So adding a new sensor is just implementing these APIs, the implementation is called LLCD (Low Level Camera Driver).

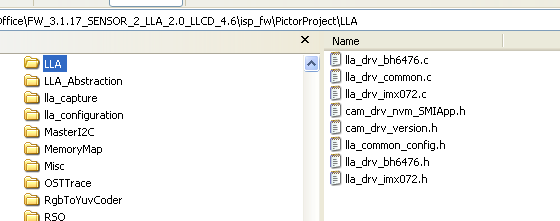
As shown in above diagram, Sensor specific changes are abstracted by LLA and all sensor specific changes implemented in LLCD. Generic FW used these API to configure and control the sensor. Further there are set of platform specific services e.g. timers, I2C read/write functionality, FSC/FEC notification etc… required by LLCD are provide by means of set of callback APIs.

Refer to LLCD documentation /4/ for complete description of low level APIs, their syntax, input output parameters.

This section provides details about the LLCD implementation, important data structure and arrays that have to be modified for integrating a new sensor, taking IMX072 as a reference example.

### 5.2.1 LLCD Directory Structure

‘PictorProject/LLA’ contains sensor specific code; image below shows the directory structure for IMX072 sensor LLCD.



As described earlier, LLCD abstract sensor module specific details from rest of the imaging stack.

In LLCD there are sensor specific, AF driver specific files. In IMX072 LLCD following are the key source files:

* lla\_drv\_common.c:

This file contains top level implementation of LLCD APIs and interface with the sensor specific implementation.

* Lla\_drv\_imx072.c/.h:

This file contains IMX072 sensor specific changes, defines IMX072 sensor init sequece, specify the different sensor modes, I2C settings for different modes, PLL configuration etc …

* Lla\_drv\_bh6476.c/.h:

IMX072 sensor module containf BH6476 AF driver IC. This file contains LLA lens API code specific to this IC.

For integrating a new sensor, it is required to add sensor and AF driver specific file into LLA directory structure.

Next sections will describe the content of these sensor and AF driver specific files in detail.

### 5.2.2 Sensor Files

IMX072 LLCD contains lla\_drv\_imx072.c/.h files that contain sensor specific code and configuration. This section provide details about these files

**Important Data structure:**

* Output modes:

LLCD API export the sensor modes supported by LLCD. These modes indicate the set of possible sensor configurations.

Each mode specify

* + Window of interest.
  + Output size.
  + Maximum Frame rate in that mode.
  + Output format to be used.
  + Use cases in which this mode can be used.

Following are the modes defined by IMX072 LLCD. These modes are evaluated from the use case and digital zoom to be supported.

|  |
| --- |
| #IMX072 Sensor modes:  /\*Suported Camera output modes for Sony IMX072 camera\*/  CAM\_DRV\_SENS\_OUTPUT\_MODE\_T g\_LLA\_IMX072\_OutputModes[] =  {  // CAM\_DRV\_USAGE\_MODE\_VF, 0  // CAM\_DRV\_USAGE\_MODE\_AF, 1  // CAM\_DRV\_USAGE\_MODE\_STILL\_CAPTURE, 2  // CAM\_DRV\_USAGE\_MODE\_NIGHT\_STILL\_CAPTURE, 3  // CAM\_DRV\_USAGE\_MODE\_STILL\_SEQ\_CAPTURE, 4  // CAM\_DRV\_USAGE\_MODE\_VIDEO\_CAPTURE, 5  // CAM\_DRV\_USAGE\_MODE\_NIGHT\_VIDEO\_CAPTURE, 6  // CAM\_DRV\_USAGE\_MODE\_HQ\_VIDEO\_CAPTURE, 7  // CAM\_DRV\_USAGE\_MODE\_HS\_VIDEO\_CAPTURE, 8// high frame rate video  // HS:8 | HQ:7 | NIGHT\_VIDEO:6 | Video:5 | Still\_seq:4 |Still Night:3 | Still\_capture:2|AF:1| VF:0,  {/\*FFoV\*/  {LLA\_SENSOR\_SONY\_IMX072\_FFOV\_X, LLA\_SENSOR\_SONY\_IMX072\_FFOV\_Y},  {LLA\_SENSOR\_SONY\_IMX072\_FFOV\_X, LLA\_SENSOR\_SONY\_IMX072\_FFOV\_Y},  CAM\_DRV\_SENS\_FORMAT\_RAW10,  0x1FB, // 111111011 only still capture allowed  0  },  ………  ………  ………      /\*H1700xV1276\*/  { // HS | HQ | NIGHT\_VIDEO | Video | Still\_seq |Still Night | Still\_capture| AF| VF,  { 1700, 1276 },  { 1700, 1276 }, //  CAM\_DRV\_SENS\_FORMAT\_RAW10,  0x15A, // 101011010  0  },  }; |

Similarly, for any new sensor these modes have to be defined.

* Sensor Initialization Sequence:

Each sensor has its own init sequence that must be programmed before any configuration to be done on the sensor.

|  |
| --- |
| # IMX072 Init Sequence  /\*PowerOn settings for IMX072 - from "IMX072ES\_RegisterSetting\_I2C\_MIPI\_2lane\_def\_rev2.0\_ST-EMP.xls"\*/  CAM\_DRV\_CAMERA\_CONFIGS\_T g\_LLA\_IMX072\_PowerOnSettings[] =  {  { 0x01, 0x03, 0x01 },  …..  …..  } |

These settings are provided by sensor vendor. And they are applied during ‘cam\_drv\_on’ LLA API.

cam\_drv\_on() -> FnSensorInitSequence()-> LLA\_IMX072\_SensorInitSequence()

Similarly, for new sensor these setting have to be defined.

* Sensor Mode settings:

As describe earlier, LLCD export set of sensor configuration as sensor modes. I2C settings for each mode are specified in form of sensor setting arrays in the LLCD.

Sensor vendor provide I2C settings for each mode to be supported. Following are the structures in which these settings have to be filled in.

a. SensorSettings\_ts:

This is the top level structure that encapsulates all the sensor settings.

|  |
| --- |
| Typedef struct  {  /// External clock frequency for selected camera  uint32 u32\_extClkFreqx100;  /// Requested CSI2 link bit rate  uint32 u32\_CSI2LinkBitRate;  ///CSI2 Data Lane Select  CAM\_DRV\_SENS\_CSI2\_LANE\_MODE\_SELECT\_E e\_CSI2\_lane\_select ;  /// Global PLL settings array.  const CAM\_DRV\_CAMERA\_CONFIGS\_T \* p\_GlobalPllModes;  /// Pointer to configuration of 1 sensor mode  const SensorModeSettings\_ts \*\* p\_SensorModeSetting;  } SensorSettings\_ts; |

‘p\_SensorModeSetting’ is an array of sensor settings one entry for each supported mode.

These settings are for a particular ‘ExtClk’ frequency, ‘CSI bitrate’ etc… If it is required to support multiple ‘ExtClk’ frequency then multiple instances of this structure has to be implemented, one for each ‘ExtClk’

E.g. for IMX072 it is required to support 1Lane and 2Lane. So we have two instances for this structure as shown below:

|  |
| --- |
| SensorSettings\_ts g\_IMX072\_SensorModeSettings[] =  {  {  //Settings for 9.6 MHz as Input Clock.  EXT\_SENSOR\_CLOCK, //u32\_extClkFreqx100  MAX\_CSI\_PLL\_OUT\_FREQ\_IMX072\_RAW10, //u32\_CSI2LinkBitRate  CAM\_DRV\_SENS\_CSI2\_LANE\_MODE\_2, //CSI2\_lane\_mode  g\_LLA\_IMX072\_GlobalPLL\_Settings\_P0, //p\_GlobalPllModes  p\_IMX072\_SensorModesArray\_P0, //  },  {  //Settings for 9.6 MHz as Input Clock.  EXT\_SENSOR\_CLOCK, //u32\_extClkFreqx100  MAX\_CSI\_PLL\_OUT\_FREQ\_IMX072\_RAW10, //u32\_CSI2LinkBitRate  CAM\_DRV\_SENS\_CSI2\_LANE\_MODE\_1, //e\_CSI2\_lane\_mode  g\_LLA\_IMX072\_GlobalPLL\_Settings\_P1, //p\_GlobalPllModes  p\_IMX072\_SensorModesArray\_P1, //  },  }; |

b. SensorModeSettings\_ts \*\*p\_SensorModeSetting

As describe earlier, there is one structure of this type for each supported mode.

|  |
| --- |
| /\*Global Structure Sensor Configurations (PLL and FDM)\*/  typedef struct  {  /// vt\_pix\_clk\_freq\_mhz  float\_t f\_VTPixelClockFrequency\_Mhz;  /// op\_pix\_clk\_freq\_mhz  float\_t f\_OPPixelClockFrequency\_Mhz;  /// op\_sys\_clk\_freq\_mhz:  float\_t f\_OPSystemClockFrequency\_Mhz;  /// vt\_pix\_clk\_period\_us  float\_t f\_VTPixelClockPeriod\_us;  /// Mode specific value for minimum line length  uint16 u16\_MiniLineLengthModeSpecific;  /// PLL Mode. If in some PLL depends on Mode selected.  const CAM\_DRV\_CAMERA\_CONFIGS\_T \* p\_PllModes;  /// Sensor Mode configurations  const CAM\_DRV\_CAMERA\_CONFIGS\_T \* p\_FDModes;  } SensorModeSettings\_ts; |

Mode specific configuration includes:

*f\_VTPixelClockFrequency\_Mhz*: Pixel clock frequency from sensor in this mode.

*f\_OPPixelClockFrequency\_Mhz* : Output pixel clock frequency, will be different from above if sensor perform de-rating internally.

*p\_PllModes*: This specifies if any mode specific PLL configurations to be applied.

*p\_FDModes*: Sensor I2C settings for this particular mode.

### 5.2.3 AF Driver Files

A typical camera module includes Sensor, AF Driver and OTP (NVM) data.

Customer may choose different AF driver and OTP mechanism.

LLA has following four API for lens movement:

* cam\_drv\_get\_curr\_lens\_pos
* cam\_drv\_lens\_move\_to\_pos
* cam\_drv\_lens\_measure\_pos
* cam\_drv\_lens\_get\_status

Implementation of these APIs depends on AF driver IC. It’s advisable to abstract LLCD implementation of these APIs.

For IMX072 LLCD, AF driver changes are abstracted in ‘*lla\_drv\_bh6476.c/.h*’

Similar implementation would be needed for new AF driver integration.

### 5.2.4 OTP/NVM Detail

Typical camera module includes Sensor, AF Driver and OTP (NVM) data. Customer may choose different AF driver and OTP mechanism.

OTP or NVM of sensor module contains module specific data e.g. module specific tuning data for lens shading correction, white balance tuning data etc. This data is then used by STE 3A algorithm for sensor tuning activities.

Sensor module may have this information in any custom format but STE 3A algorithm expects these data to be present in a specific format defined by SMIA++.

|  |
| --- |
| SMIA++ NVM MAP defined in ‘cam\_drv\_nvm\_SMIApp.h’ |

It’s a LLCD responsibility to convert OTP data into SMIA++ format as expected by 3A algorithms.

Some module may not have these data available as part of NVM, in such cases STE imaging stack provide a mechanism to have these data available to 3A algorithm via external files.

|  |
| --- |
| # Enabling NVM from file feature  To enable 3A algorithm to read NVM data from file, following modification has to be done:   1. LLCD should report about absence of NVM data by setting   *p\_camera\_details->p\_nvm\_details->size = 0*   1. Following flag in Imgconfig.h file must be set:   *define IMG\_CONFIG\_NVM\_FROM\_FILE IMG\_CONFIG\_TRUE* |

## 5.3 MMIO Driver

MMIO driver control:

* XSHUTDOWN Signal.
* Voltage supply.
* ExtClk frequency.

Depending upon the board and module design, changes may have to be done in MMIO drivers e.g. if XSHUTDOWN mapped to different ISP GPIO, if we need to change the ExtClk frequency etc…

### 5.3.1 XSHUTDOWN Signal.

XSHUTDOWN signal act as reset signal for sensor. As per current design XSHUDOWN signal is derived by ISP FW via ISP GPIOs.

U8500 system GPIOs can be configured to work as normal GPIOs or acting as GPIO for some IP. There are eight U8500 GPIOs that can be configured to work as ISP GPIO. In order to control XSHUTDOWN signal via ISP FW, Sensor XSHUTDOWN pin must be connected to one of ISP controllable GPIO.

STE reference design (HREF) uses U8500 GPIO141 that mapped to IP\_GPIO2.

If some custom board has different mapping for XSHUTDOWN pin, it’s required to do modification in MMIO driver (kernel).

|  |
| --- |
| #Shutdown configuration in MMIO driver  In file ‘*kernel/arch/arm/mach-ux500/board-mop500-mmio.c*’  Need to define different GPIO under following GPIO pin configuration.  static pin\_cfg\_t xshutdown\_host[] = {  GPIO141\_GPIO,  GPIO142\_GPIO  };  static pin\_cfg\_t xshutdown\_fw[] = {  GPIO141\_IP\_GPIO2,  GPIO142\_IP\_GPIO3  };  static pin\_cfg\_t xshutdown\_disable[] = {  GPIO141\_GPIO | PIN\_OUTPUT\_LOW,  GPIO142\_GPIO | PIN\_OUTPUT\_LOW  }; |

### 5.3.2 ExtClk frequency

As per current design, ExtClk (Main Clock) to the sensor is originated from 38.4MHz clock source. And the clock frequency can be reduced by applying a 1/2 or 1/4 devisor.

On HREF main clock to sensor is 9.6 MHz (38.4/4) and obtained by dividing the main clock by factor of 4.

|  |
| --- |
| #Changing sensor main clock frequency  In file ‘*kernel/arch/arm/mach-ux500/clock-db8500.c*’, in function ‘clkout0\_enable’  static int clkout0\_enable(struct clk \*clk)  {  …….  …….  r = regulator\_enable(clk->regulator);  if (r)  goto regulator\_failed;  r = prcmu\_config\_clkout(0, PRCMU\_CLKSRC\_CLK38M, **4**); 🡨 Main CLK divisor  if (r)  ……  ……  } |

# 6. Validation

This section provides details about STE Imaging validation environments.

It will cover the basic test cases to run to ensure the sensor integration is completed. More exhaustive test suits detail can be found in /5/ and /6/.

## 6.1 ISP FW Validation

STE ITE-NMF test suit is used for validating sensor integration/adaptation at ISP FW level. ITE-NMF environment is available as part of STE release.

As described in section *4.1.1,* validation environment are compiled out by default. '*BUILD\_VALID*’ and ‘*BUILD\_ITE\_OMX\_USE\_CASES*’ must be set to ‘true’ in Dirs.u8500.mk file to include them in compilation.

Refer /5/ present at <path>/vendor/st-ericsson/multimedia/imaging/ite\_nmf/docs

Refer this for more detail:

* On compiling/running ITE-NMF environment.
* On various test cases available at ITE-NMF.

## 6.2 OSI Components Validation

STE MMTE test suit is used for validating sensor integration/adaptation at OSI level. MMTE environment is available as part of STE release.

As described in section *4.1.1,* validation environment are compiled out by default. '*BUILD\_VALID*’ and ‘*BUILD\_ITE\_OMX\_USE\_CASES*’ must be set to ‘true’ in Dirs.u8500.mk file to include them in compilation.

Refer /6/ for more detail on compiling/running MMTE environment.

# 7. Open Points

1. ITE-NMF Test case to be updated to validate new sensor integration.
2. Do\_kanna.ite is a sensor dependent MMTE script, changes in the script to be identified.